OFF-POLICY REINFORCEMENT LEARNING FOR BIPEDAL ROBOT LOCOMOTION

An Undergraduate Honors Thesis
Submitted to the Department of Mechanical Engineering
In Partial Fulfillment of the Requirements
For Graduation with Distinction at The Ohio State University

Tom Ballas
April, 2021

Advisor: Dr. Ayonga Hereid

Abstract

Reinforcement Learning (RL) is a developing learning-based approach that has shown potential to be instrumental in the programming of bipedal robots. RL allows robots to learn ideal behaviors through many iterations of trial and error. The objective of RL is to learn an optimal policy, which is a mathematical function that takes in an agent's state as input and outputs an optimal action. Traditional RL approaches have been on-policy in nature meaning they attempt to improve the policy that is used to make decisions. However, the inability of these methods to utilize training data not generated from the policy often leads to a data inefficient training process and results in policies that generalize poorly to unseen data. These limitations have led researchers to pursue off-policy algorithms. One such method, Deep Deterministic Policy Gradients (DDPG), utilizes an experience buffer and multiple neural networks to learn ideal actions in environments with continuous actions spaces. While DPPG methods have been successful in some robotic control problems, they are prone to converge to suboptimal solutions when high reward actions are not discovered early in training. In particular, when used for bipedal walking, they have been unable to learn policies that produce stable walking movements. In this work, a revised DDPG RL approach is proposed that incorporates physical insights of robot walking and utilizes previously collected actions. The proposed framework is tested on the RABBIT robot model in OpenAI Gym with the MuJoCo physics engine. This approach is successful in training RABBIT to complete tasks like walking at desired velocities and walking up hills of varying grade. This work exhibits the value of utilizing external data in developing data efficient and generalizable RL approaches.

Acknowledgements

I would first like to thank my advisor, Dr. Ayonga Hereid. His guidance, support, and advice were instrumental in the success of this project. I appreciated how he provided me the flexibility to explore many possible routes for the project while also making sure the project stayed on track to be completed. I am grateful that he dedicated time each week to discuss this project and provide advice and feedback.

I would also like to thank Guillermo Castillo for volunteering his time to help me with this project. I began this project with limited knowledge of Reinforcement Learning and Robotics, and Guillermo was very supportive in helping me learn this material. I am also appreciative that he provided feedback on the project each week, and the project certainly would not have been a success without his help.

I also owe thanks to Dr. Stephanie Stockar and Dr. Hanna Cho. Their feedback on my research presentations and instruction on becoming an effective researcher was valuable in preparing my thesis. I would also like to thank Dr. Manoj Srinivasan for agreeing to be a member of the committee for my oral defense and providing valuable feedback on the project.

Table of Contents

List of Figures

List of Tables

Chapter 1. Introduction

1.1) Background of Reinforcement Learning for Bipedal Robots

Stable bipedal locomotion presents a challenging problem for robotics researchers for a variety of reasons including the complexity of high dimensional models, unilateral ground contacts, and nonlinear and hybrid dynamics (Castillo et al., 2019). Most existing control methods for bipedal locomotion rely on accurate physical models of a system, but these models can be hard to derive for complex robotic systems. In addition, biped walking dynamics include contact and collision between the robot and the ground, which makes precise modeling of the dynamics difficult (Castillo et al., 2019). Due to these challenges, there is growing interest in using Reinforcement Learning (RL) to obtain effective control policies for bipedal robots as these methods do not require analytic models of the robot (Castillo et al., 2019; Kim et al., 2017; Xie et al., 2019).

1.2) Reinforcement Learning Overview

RL allows an agent to learn ideal behaviors in its environment through many iterations of trial and error (Zhu et al., 2019). The objective of RL is to train a mathematical function, called a policy, which takes in the agent's state as input and outputs an optimal action. During training, the agent receives rewards based on how close the action provided by the policy is to the desired action. Over many iterations, the policy is updated based on these rewards to better predict desirable actions (Agarwal & Norouzzi, 2020). In most modern day RL algorithms, policies take the form of deep neural networks (DNNs) where the weights and biases of the network can be adjusted via some optimization algorithm to change the behavior. Figure 1 illustrates the basic RL paradigm with π as policy and θ as DNN weights.

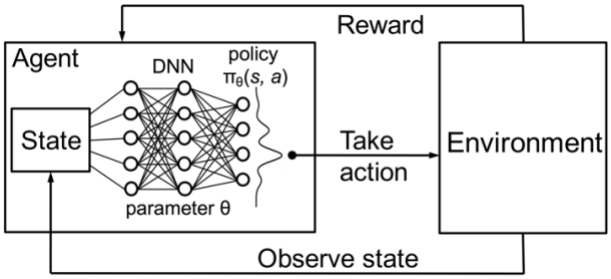


Figure 1*: Diagram of basic facets of RL (Li, 2018)*

1.2.1) Reinforcement Learning Objective

An RL problem can be defined mathematically using Markov Decision Processes (MDPs). An MDP is a six tuple, $< S, A(s), R(s,a), P(s'|s,a), p_0, \gamma >$, that consists of a set of environment states S, a set of possible actions from each state $A(s)$, a real valued reward function $R(s,a)$, a transition model $P(s'|s,a)$, a starting state distribution p_0, and a discount factor γ. Using MDPs, a trajectory can be defined as a sequence of states and actions of length H, given by $\tau = (s_0, a_0, \dots \dots s_H, a_H)$, where H may be infinite. The objective of RL is to train a policy $\pi(a_t|s_t)$ which will provide the actions that will maximize the expected return over some trajectory. The return over some trajectory, where r_t is the reward at each timestep and γ is a discount factor, can be written as:

$$R(\tau) = \sum_{t=0}^{H} \gamma^t r_t \tag{1}$$

The probability of a given trajectory under a policy can be written as:

$$P(\tau|\pi) = p_o(s_o) \prod_{t=0}^{H-1} P(s_{t+1}|s_t, a_t)\pi(a_t|s_t) \tag{2}$$

Utilizing both the equations for the return and the probability of a trajectory, we can determine the expected return $J(\pi)$ which is shown in (3).

$$J(\pi) = \int_{\tau} P(\tau|\pi)R(\tau) \tag{3}$$

The central optimization problem for RL is simply to find the policy π^*, that maximizes $J(\pi)$. $J(\pi)$ is often referred to as the RL objective (Levine et al., 2020).

1.2.1) Reinforcement Learning Algorithms

Researchers have developed a wide variety of methods for training an optimal policy. One approach is to directly optimize the RL objective (Levine et al., 2020). This is known as policy optimization, and a common practice for accomplishing this is to estimate the gradient of $J(\pi)$. With this gradient $\nabla_\theta J(\pi_\theta)$, a policy, in the form of a neural network with parameters θ, can be trained to maximize the RL objective using gradient ascent as shown:

$$\theta_{k+1} = \theta_k + \nabla_\theta J(\pi_\theta). \tag{4}$$

The derivation for the gradient of the expected return $\nabla_\theta J(\pi_\theta)$ is shown in Figure 2.

$$\begin{aligned}
\nabla_\theta J(\pi_\theta) &= \nabla_\theta E_{\tau\sim\pi_\theta}[R(\tau)] \\
&= \nabla_\theta \int_{\tau} P(\tau|\theta)\, R(\tau) \\
&= \int_{\tau} \nabla_\theta P(\tau|\theta)\, R(\tau) \\
&= \int_{\tau} P(\tau|\theta)\, \nabla_\theta log\ P(\tau|\theta) R(\tau) \\
&= E_{\tau\sim\pi_\theta}[\nabla_\theta log\ P(\tau|\theta) R(\tau)] \\
\nabla_\theta J(\pi_\theta) &= E_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta log \pi_\theta(a_t|s_t) R(\tau)\right]
\end{aligned}$$

Figure 2*: Derivation for Gradient of Expected Return (Achiam, 2018)*

With this derivation, trajectories (sequences of states and actions) generated using the policy and the resulting rewards can be utilized to estimate $\nabla_\theta J(\pi_\theta)$ and update the policy. A visualization of the policy update process for policy optimization is shown in Figure 3, and

pseudocode for a basic policy gradient method can be found in Appendix A: Pseudocode for Popular RL Algorithms.

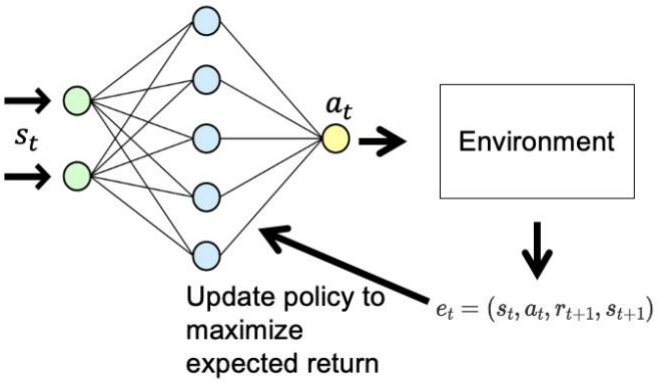


***Figure 3**: Policy Updates for Policy Optimization RL Algorithms*

It should be noted that there are also policy optimization methods that do not utilize the gradient of the policy, but all of these methods implement the basic idea of attempting to directly maximize the RL objective. Since policy optimization methods attempt to directly maximize the expected return, trajectories utilized to update the policy must be generated from the current policy. RL algorithms with this restriction are designated as on-policy. Although on-policy RL algorithms are often stable and reliable, the limited data they can use for updates makes them data inefficient (Achiam, 2018).

Another method for optimizing the RL objective is to develop a state-action value function $q_\pi(s, a)$ for a policy π (Levine et al., 2020). This function can provide the expected cumulative reward of taking an action from some state and then following π thereafter. In RL, this state-action value function is referred to as the Q-function. Mathematically, the Q-function can be defined as:

$$q_\pi(s, a) = \mathrm{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k \mathrm{R}_{t+k+1} \middle| \mathrm{S}_t = s, \mathrm{A}_t = a\right]. \quad (5)$$

The objective of this RL problem can just be to find the Q-function $q_*(s, a)$ that gives the largest expected reward for a policy π for each state-action pair. This can be accomplished by utilizing the Bellman optimality equation, which is shown in (6), with γ as a learning rate between 0 and 1 and $\mathrm{R}_{\mathrm{t+1}}$ as a reward.

$$\mathrm{q}_*(\mathrm{s},\mathrm{a}) = \mathrm{E}\left[\mathrm{R}_{\mathrm{t+1}} + \gamma \max_{\mathrm{a}'} \mathrm{q}_*(\mathrm{s}',\mathrm{a}')\right]. \tag{6}$$

To develop an optimal Q-function, the objective becomes to minimize the difference between $q_\pi(s, a)$ and the right hand of the Bellman equation for each state action pair. A neural network with parameters ϕ can be trained to approximate the Q-function by minimizing the following expression for each state action pair:

$$\mathrm{q}_\phi(\mathrm{s},\mathrm{a}) - \left(\mathrm{R}_{\mathrm{t+1}} + \gamma \max_{\mathrm{a}'} \mathrm{q}_\phi(\mathrm{s}',\mathrm{a}')\right). \tag{7}$$

Once the optimal Q-function is developed, the optimal policy is to simply return the action that returns the highest value from the Q-function. In similar fashion to policy gradient methods, trajectories can be generated, and the resulting rewards can be utilized to update the neural network. However, in contrast to policy gradient methods, these trajectories do not need to be generated by the current policy. This feature makes Q-learning an off-policy RL algorithm. Off-policy RL algorithms often utilize an experience buffer to be more sample efficient. This buffer contains a number of experience tuples in the form (s_t, a_t, s_{t+1}, r_t) that can be sampled to update the policy. A visualization of Deep Q-learning utilizing an experience buffer is shown in Figure 4, and pseudocode for Deep Q-Learning can be found in Appendix A: Pseudocode for Popular RL Algorithms.

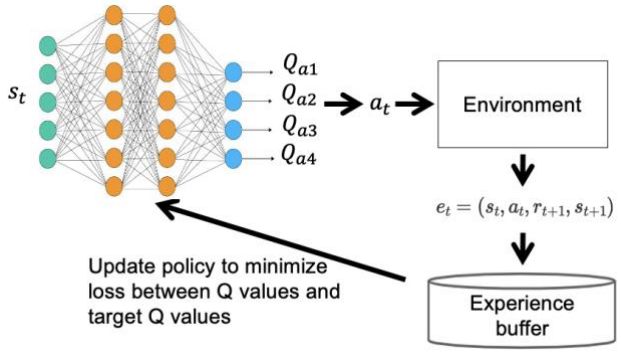


***Figure 4**: Deep Q-learning using Experience Buffer*

As is shown in the figure above, the network returns a value from the Q-function for each discrete action. This makes it very challenging to utilize deep Q-learning in problems with continuous actions spaces like robotics. However, there are modern approaches that utilize the foundations of Q-learning to develop off-policy methods that can solve problems with continuous actions spaces. These approaches are discussed in-depth in the Review of Relevant Literature. Another issue with traditional Q-learning is the target values generated from the right side of the Bellman equation are dependent on the same parameters ϕ that are being updated to train the policy. This can make the training of a network with Q-learning an unstable process. Many modern approaches utilize a target network which is a time-delayed copy of the original network that can add stability to training (Levine et al, 2020). Target networks are discussed in-depth in the Implementing DDPG Algorithm with Offline Data section of Chapter 2.

In addition to policy gradients and Q-learning, the RL objective can also be achieved by utilizing or learning a model of the environment. A model in this case is a function that predicts state transitions $P(s'|s,a)$ and rewards $R(s,a)$. Having a model allows the agent to plan ahead to see the potential results of its choices of actions. Based on these results, it can develop a policy to choose the optimal actions. Model-based RL is very sample efficient, but a ground-truth model of the environment is not available to an agent in most RL settings (Achiam, 2018).

This means that model based RL approaches must learn a model from experience. The main issue with this approach is bias in the learned model can be exploited by the agent (Achiam, 2018). This means the agent often performs well with respect to the model but poorly in the real environment. In addition, model based RL algorithms often require far more compute power than approaches not utilizing a model (model-free methods). These challenges have prevented model-based methods from becoming as popular as model-free methods. However, model-based RL is an active area of research, and recently developed model-based approaches have been successful on many continuous control benchmarks (Janner et al., 2019).

Given the benefits and limitations of policy optimization, Q-learning, and model-based learning, many modern RL approaches utilize aspects of each method. There are some approaches that utilize models to optimize a policy or learn a Q-function. In addition, actor-critic algorithms attempt to combine Q-learning and policy optimization by using a parametrized policy and value function. These methods are discussed in more detail in the Review of Relevant Literature.

1.3) Review of Relevant Literature

In this section, the current status of research in using RL for bipedal robots is described and analyzed. In particular, the drawbacks of various methods are discussed to provide context and motivation for this project.

1.3.1) On-Policy Reinforcement Learning for Bipedal Locomotion

Most approaches utilizing RL for bipedal locomotion employ policy gradient methods to find policies to map some state space to an action space for continuous walking motion. The stability and reliability of policy gradient methods along with their ability to operate in

environments with large continuous action spaces makes them a natural choice for bipedal locomotion. However, RL methods for bipedal robots have often led to unnatural walking motions that are not applicable to robots as they do not consider the underlying physics of bipedal walking. As a result, Castillo et al. (2019) developed a RL method for achieving bipedal walking that incorporated physical insights of bipedal walking to a policy gradient method. They utilized insights from Hybrid Zero Dynamics (Westervelt et al., 2007), which is a powerful tool for bipedal control with local stability guarantees of the walking limit cycles. An illustration of their learning control structure is shown in Figure 5.

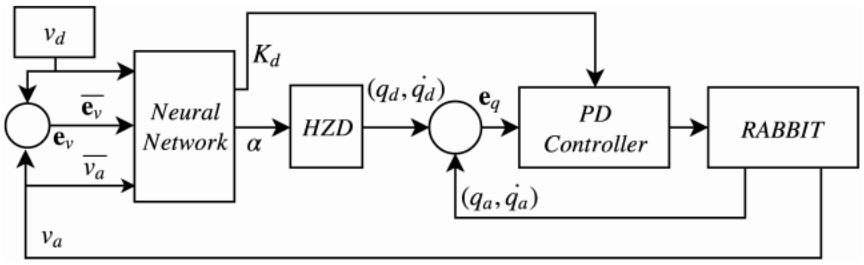


Figure 5*: Robot Learning Control Structure used in Castillo et al. (2019)*

Instead of directly mapping robot states to control actions, their proposed policy mapped a reduced order of the robot's state $(e_v, \overline{v_a}, v_d)$ to a set of coefficients α of the Bezier polynomials that defined the trajectory of the actuated joints. These trajectories were then tracked using an adaptive PD controller. This method was tested on the RABBIT robot in simulation and was able to track desired velocities while maintaining stable walking.

1.3.2) Need for Off-Policy and Offline Reinforcement Learning

Despite the success of this approach, the on-policy/online nature of the policy gradient algorithm creates some challenges. In Levine et al. (2020), the potential drawbacks of on-policy algorithms are discussed. When the policy in an on-policy approach is updated, the samples used to update this policy are discarded since future updates will require data generated from the

most recently learned policy. This constant discarding of data causes these methods to be data inefficient and require large amounts of data to determine an optimal policy. In the article, the authors discuss how the success and generalizability of modern machine learning methods can be largely attributed to their utilization of large and diverse previously collected datasets. The restriction of only utilizing training data from the most recent policy has limited on-policy RL algorithms from reaping these benefits. The authors also discuss some alternatives to on-policy RL including off-policy and offline RL. Figure 6 illustrates the key differences between online (on-policy), off-policy, and offline RL.

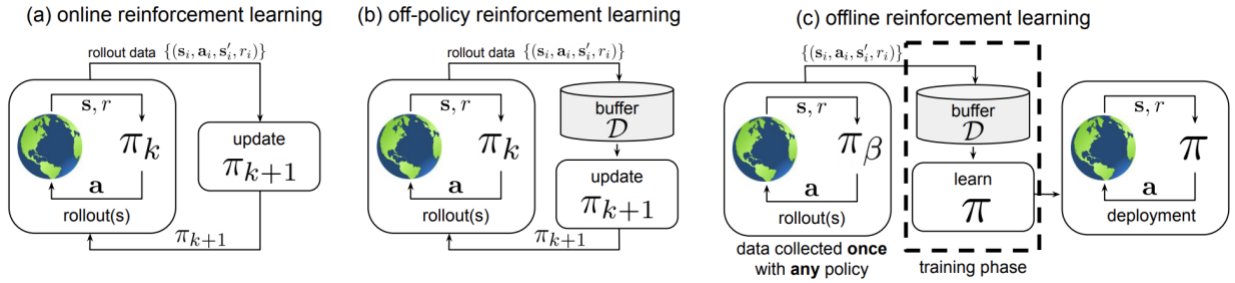


***Figure 6**: Comparison of Online, Off-Policy, and Offline RL (Levine et al., 2020)*

For offline RL, the algorithm is provided with a static dataset of transitions, $D = \{(s_t^i, a_t^i, s_{t+1}^i, r_t^i)\}$, and must learn the best policy using this dataset. However, there are major algorithmic challenges associated with offline RL. Training and evaluating a policy on different distributions can result in distribution shift. Another approach is implementing off-policy algorithms which can learn from off-policy data, but also require some online interaction to limit distribution shift. Using off-policy data is particularly useful in robotics. For example, RL methods in robotic grasping must be able to generalize to a wide range of objects. Utilizing a large and diverse dataset in training is essential in producing this generalizability, and researchers have developed off-policy methods for grasping that generalize to objects unseen in training (Kalashnikov et al., 2018).

1.3.3) DDPG for Off-Policy Reinforcement Learning

One major challenge with utilizing off-policy algorithms for bipedal locomotion is many of these algorithms, like Deep Q-learning, can only handle discrete, low-dimensional action spaces. This limitation has led researchers to explore potential off-policy methods that could handle high-dimensional and continuous action spaces. One such method developed in Lillicrap et al. (2015) is known as Deep Deterministic Policy Gradients (DDPG). DDPG attempts to concurrently learn a Q-function and a policy. Off-policy data from an experience buffer is utilized to learn the Q-function, and the Q-function is used to learn the policy. DDPG has been successful at completing many classic physics tasks like the cartpole swing-up, dexterous manipulation, and car driving. Despite these successes, DDPG often fails in many legged locomotion problems including obtaining stable walking motion for the RABBIT robot. DDPG often suffers from instability as it is sensitive to hyperparameters and is inclined to converge to suboptimal solutions. In Matheron et al. (2019), a study was conducted that analyzed the common issues with DDPG. The authors discuss how in cases of sparse reward, if high rewards are not initially found, the actor may drift to a poor policy. The sparsity of the rewards can cause the Q-function to converge to a piecewise constant function with relatively flat gradients which will cause the actor to barely update. This process is illustrated in Figure 7.

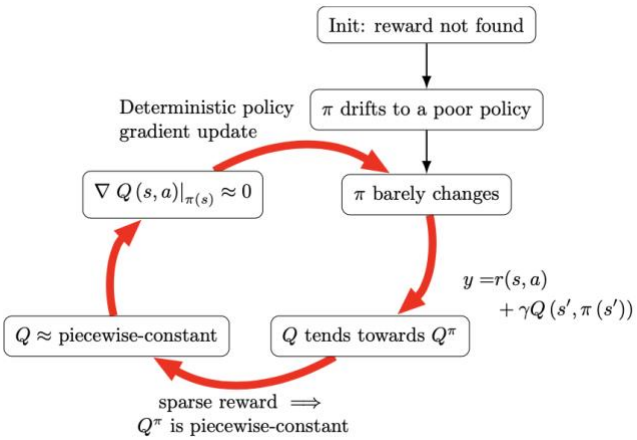


***Figure 7**: Process for Convergence of DDPG to Suboptimal Policy (Matheron et al., 2019)*

This process is very challenging to avoid in the case of bipedal locomotion. Due to the complexity imposed by high-dimensional robot models, an untrained policy is very unlikely to generate meaningfully high rewards to prevent the suboptimal convergence process from occurring. However, due to the off-policy nature of DDPG, the untrained policy does not need to be the policy generating actions. Any relevant offline data can be utilized to generate actions with high enough rewards to prevent the Q-function from having no to little gradient. This thesis focuses on developing methods that utilize offline data to allow DDPG to be functional for bipedal locomotion.

1.4) Objective of Thesis

The objective of this project is to design a data efficient and generalizable RL approach that will allow for the training of RL policies for a variety of robot locomotion tasks including velocity tracking and sloped terrain movement. As is shown in the relevant literature, on-policy RL approaches are data inefficient, and their inability to utilize diverse datasets prevents them from generalizing well to unseen data. Off-policy methods often work best with low-dimensional action spaces, and even off-policy algorithms that are designed for higher dimension

spaces, like DDPG, can converge to suboptimal solutions. However, utilizing previously collected data has potential to prevent this issue and allow DDPG to develop optimal policies for problems in bipedal locomotion. In this thesis, an approach is proposed that incorporates physical insights of robot walking and previously collected actions into the DDPG algorithm to create a data efficient RL approach that can obtain generalizable policies for bipedal locomotion.

1.5) Research Significance

Using RL for intelligent adaptive control for walking has shown potential to greatly impact the next generation of robotic technology, including humanoid robots for space exploration and search and rescue missions, increased autonomy in robotics, and wearable robotic devices that improve the quality of life for the mobility-impaired population. Utilizing offline data to improve data efficiency and policy generalizability may allow RL to achieve this potential. In addition, this research has potential to have impacts beyond just bipedal locomotion. RL approaches have been attempted in the fields of autonomous driving, healthcare, and natural language with mixed success. However, similar to robotics, the data inefficiency and inability to produce policies that generalize to many tasks has limited their effectiveness and impact. Therefore, using offline data to curb these limitations in the setting of bipedal locomotion has potential to profoundly impact a wide assortment of fields beyond robotics.

1.6) Overview of Thesis

This thesis has five chapters. Chapter 2 examines the process of designing the RL approach for bipedal locomotion. This consists of selecting a robot and environment, defining the state, action, and reward space, implementing the DDPG algorithm with offline data, and

developing procedures for method evaluation. Chapter 3 discusses evaluating the RL method in allowing a bipedal robot to walk at desired velocities. This includes refining the approach to produce a policy that converges to a high reward, evaluating the effects of the offline data, and comparing the final result to policies from current on-policy approaches. The purpose of this chapter is to show that the proposed off-policy method can train a policy that exhibits comparable performance on velocity tracking as on-policy approaches but does so using far fewer training episodes. Chapter 4 exhibits how data from velocity tracking can be used to train the robot to walk up hills of varying grade. The purpose of this chapter is to illustrate that an off-policy approach can utilize off-policy data to complete tasks that are currently not realized by on-policy methods. Chapter 5, the conclusion, summarizes the key contributions of this thesis, discusses additional applications, and proposes possible future directions of study.

Chapter 2. Designing Off-Policy RL Method for Bipedal Locomotion

The methodology for designing this off-policy RL method was partitioned into sections for determining robots to test, defining the state, action, and reward space, implementing the DDPG algorithm with offline data, and developing procedures for method evaluation.

2.1) Selecting Robot and Environment

As a starting point for the proposed method, the model of the robot RABBIT was utilized. Despite its simple mechanical structure, RABBIT still provides a suitable representation of biped locomotion. In addition, its use in developing on-policy RL methods provides a baseline for comparison for the developed off-policy method. The RABBIT robot is a five-link, planar underactuated bipedal robot, and the five links of the robot correspond to the torso, right thigh, right shin, left thigh and left shin (Castillo et al., 2019). The robot has point feet and four actuated joints, two in the hip joints and two in the knee joints (Castillo et al., 2019). Table 1 contains a description of the length, mass, and inertia of each link of the robot.

	Torso	Femur	Tibia
Length [m]	0.63	0.4	0.4
Mass [kg]	12	6.8	3.2
Inertia [$kg*m^2$]	1.33	0.47	0.2

Table 1*: Description of length, mass, and inertia of each link of RABBIT Robot (Castillo et al., 2019)*

A schematic of the RABBIT robot with its joint cordinates, $(q_t, q_{sh}, q_{sk}, q_{nsh}, q_{nsk})$, is shown in Figure 8.

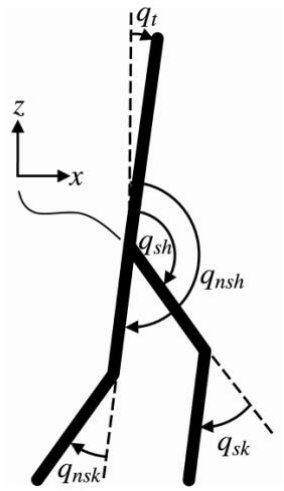


Figure 8*: Schematic of RABBIT robot with joint coordinates (Castillo et al., 2019)*

For testing the RABBIT robot, a customized environment for RABBIT built in OpenAI Gym (Brockman et al., 2016) was used. The environment was simulated using the MuJoCo (Todorov et al., 2012) physics engine. For training a robot in simulation, the robot takes an action every 2 milliseconds, and the simulation returns the state of the environment. The process of taking an action and returning a new state is defined as a simulation step. During training, these steps can be continuously taken until the robot reaches either a stopping state (falls over) or a maximum number of steps. This process is defined as an episode. Many episodes are simulated to provide data to train the robot.

2.2) Defining State, Action, and Reward Space

Defining the state space for an RL approach involves determining what data will be passed to the policy as its input. This data can take a wide variety of forms including the actual and desired values of the robot velocity, height of hip joint, distance between feet, and torso angular velocity. Adding features to the state space provides the policy with more information to

determine optimal actions, but too large of a state space may cause the policy to overfit to irrelevant attributes.

Effectively defining the state space can be highly dependent on the objective of the approach. For the first test of this project, the objective was to have the robot track various desired velocities. The value of the desired velocity was uniformly sampled from a continuous space interval from 0.6 to 1.6 m/s. A common approach for defining the state space for RABBIT involves using the desired and current velocity of the robot's hip (Castillo et al., 2019). However, while maintaining the desired velocity is important, in order to accomplish a stable walking motion, a robot must maintain its hip height to prevent itself from falling. Therefore, the hip height can help provide insight to the policy when the robot is about to fall. In addition, the current velocity may not be ideal to include in the state space due to the complex dynamics of the walking motion. This complex dynamics makes it impossible to guarantee a good tracking performance for the current velocity of the robot along the x-axis. Instead, the average velocity of one walking step of the robot (approximately 200 simulations steps) was used. The inputs to the policy in the implementation were the desired velocity, the average hip's velocity (v_a) of the robot for the last 200 simulation steps, and the current hip height (h). The state space is visualized on the RABBIT robot in Figure 9.

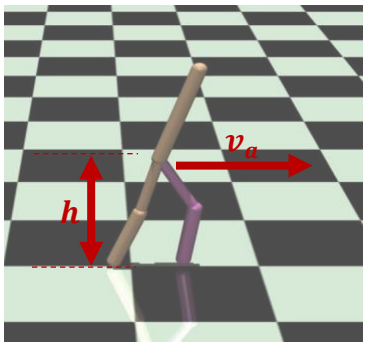


***Figure 9**: Visualization of State Attributes on RABBIT Robot*

Defining the actions involves determining what the policy will return as output. Typically, these actions are simply defined as torques for individual joints of the robot. However, this approach can result in unnatural robot movements as it does not account for the underlying physics of bipedal walking. Therefore, for defining the actions, an approach that incorporates physical insights of bipedal locomotion from Castillo et al. (2019) was used. As was discussed in the Review of Relevant Literature, this approach utilizes insights from Hybrid Zero Dynamics to enable stable walking motion. Each of the four actuated joints for the RABBIT robot are defined by a 5^{th} order Bezier polynomial that defines the trajectory of the joint. The output of the policy is defined as the twenty coefficients from the four polynomials. These coefficients are utilized by the adaptive PD controller to determine the torques of each individual joint. These torques are then provided as actions in the simulation environment.

In addition to defining the state and action space, the reward function also must be defined. The reward function attempts to reinforce actions that are desirable with high rewards and punish actions that are undesirable with low rewards. In the case of tracking desired velocities, desirable actions are actions that cause the robot velocity to be close to the actual velocity. It is important to note that actions are defined as the output of the policy. So, in this case, the actions are the coefficients defining the robot trajectory and not the torques of each robot joint. As was previously discussed, the robot velocity was defined as the average velocity from the previous 200 simulation steps. One simple option would be to utilize the average velocity and desired velocity in calculating the reward. However, since the average velocity is from the previous 200 steps, it doesn't make sense to provide a reward to the current action based on a value that results from the previous 200 actions. Another option would be to base the reward on the current velocity instead of the average, but as was previously discussed, the

current velocity in the x-direction can be a volatile measurement due to the complexity of bipedal walking. In addition, it can be hard to see the result of a single action in just one simulation step (2 milliseconds).

A third option is to only calculate new actions every set number of steps s. This allows there to be multiple steps of some action before determining the reward. In a normal setting, this may not be ideal since taking the same actions over multiple steps may result in suboptimal behavior. However, in this case, the adaptive PD controller determines torques given the coefficients provided to it and the current state. Even if the coefficients are not being updated each step, the controller can still provide different torques since the state of the robot is changing. In addition, for stable walking, the coefficients defining the robot trajectories should stay relatively constant over the course of robot walking, so it is not critical that they change on every step. This approach does require determining the ideal number of steps in between taking each action. After testing multiple potential values, taking 10 steps between each action exhibited the best results. Using the average velocity over the previous s steps v_a and the desired velocity v_d, the reward R_v was defined as shown:

$$R_v(v_a, v_d) = \left(\frac{v_a}{v_d}\mathrm{I}(v_d \geq v_a) + \frac{v_d}{v_a}\mathrm{I}(v_a > v_d)\right)\mathrm{I}(v_a > 0) \; - \; 0.5\mathrm{I}(v_a \leq 0). \qquad (8)$$

Note that an indicator function $I(x)$, that returns 1 when x is true and 0 when x is false is used to simplify writing the reward function. Using the reward function above allows for rewards near 1 when v_a is close to v_d and rewards close to 0 when they are not. It also provides for a reward of -0.5 if the average velocity is less than or equal to 0.

2.3) Implementing DDPG Algorithm with Offline Data

The DDPG algorithm involves two main objectives: learning a Q function and utilizing the Q function to learn a policy. For learning a Q-function, a neural network called the critic is trained, while the policy is learned by a neural network called the actor. The data utilized for training these networks is sampled from a replay buffer. The replay buffer, D, contains a set number of previous experience tuples with each containing a starting state, action taken, reward received, and next state (s, a, r, s'). Since DDPG is an off-policy algorithm, the data in the replay buffer has no restrictions where it can be generated from. In most implementations of DDPG for robotics in simulation, the robot begins in some stable state and then each simulation step, takes an action based on what is provided by its policy (actor network). Based on the state and action, a reward and new state are returned. To increase exploration, small amounts of noise are added to actions returned from the policy. At each step, the (s, a, r, s') tuple is added to the replay buffer. The robot continually takes actions in its environment until it reaches a stopping state. This process can be continuously repeated to provide the replay buffer with new data until the end of training.

This approach for data generation works well if the actor is able to generate high reward actions early enough in training such that the critic is able to learn which actions generate high Q values in given states. As was discussed in the Review of Relevant Literature, if this does not occur, the algorithm is susceptible to converge to suboptimal solutions. To prevent this, the presented off-policy approach utilized an offline dataset of known actions. An illustration of the data generation process for DDPG with offline data is shown in Figure 10.

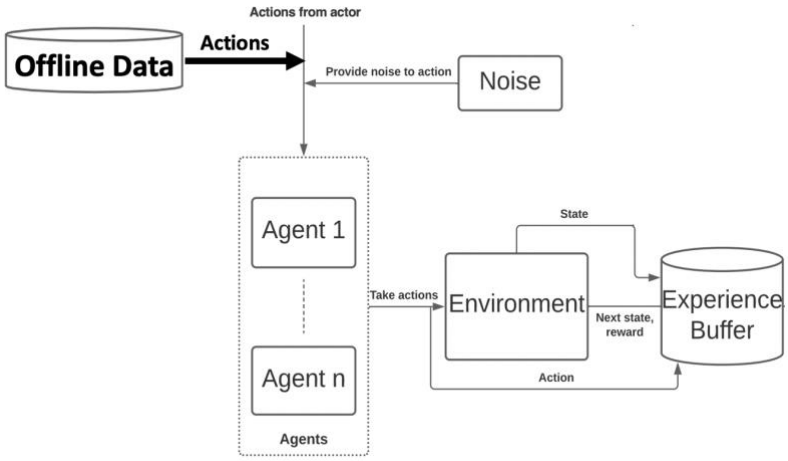


***Figure 10**: DDPG Data Generation with Offline Data*

To obtain data for velocity tracking, a program called FROST (Fast Robot Optimization and Simulation Toolkit) was used to generate data. FROST is an open-source MATLAB toolkit for modeling, trajectory optimization and simulation of hybrid dynamical systems with a particular focus in dynamic locomotion (Hereid and Ames, 2017). It can be used to generate Bezier coefficents that result in stable walking motions at various velocities for RABBIT. These coefficients are obtained via trajectory optimization using the ideal robot dynamics and do not guarantee stable walking in MuJoCo simulation or real hardware. However, this data can be helpful in training RL policies as it will likely generate actions with higher rewards than an untrained policy. In this case, ten sets of coefficents generated from FROST for velocities varying from 0.6 m/s to 1.5 m/s were used. Since there were only ten sets of coefficents, noise was added to the coefficents during training, as is shown in Figure 10, in an effort to expand the limited offline dataset.

Once there is data in the replay buffer, training of the actor and critic neural networks can begin. The actor and critic training method that was used was adopted from the original DDPG method outlined in Lillicrap et al. (2015). Complete psuedocode of this method is provided in

Appendix A: Pseudocode for Popular RL Algorithms. In this method, the objective when training the actor network is simply to learn a policy $\mu_\theta(s)$ with parameters θ that maximes the Q-value returned by the critic network $Q_\phi(s,a)$. The actor objective where the Q-function parameters ϕ are treated as constants can be shown mathematically as:

$$\max_{\theta} \underset{s\sim D}{E}\left[Q_\phi\left(s,\mu_\theta(s)\right)\right]. \tag{9}$$

The objective in training the critic is to minimize the mean-squared Bellman error (MSBE) function, which defines the difference between the network output and the result of the Bellman equation. The MSBE function $L(\phi,D)$ for a critic neural network $Q_\phi(s,a)$ with parameters ϕ and hyperparameter γ is shown in (10).

$$L(\phi,D) = \underset{(s,a,r,s')\sim D}{E}\left[\left(Q_\phi(s,a) - \left(r + \gamma \max_{a'} Q_\phi(s',a')\right)\right)^2\right]. \tag{10}$$

However, one issue with utilizing the MSBE function is the result of the Bellman equation is dependent on the same parameters ϕ that are being trained. This can make minimizing the MSBE function an unstable process. In addition, determining the maximum over actions is very challenging in settings with continuous action spaces. DDPG solves these issues by utilizing target networks which are time delayed versions of the original actor and critic. Using the hyperparameter ρ the target networks parameters ϕ_{targ} are updated as shown:

$$\phi_{targ} = \rho\phi_{targ} + (1-\rho)\phi. \tag{11}$$

Due to the delay, the target critic network $Q_{\phi_{targ}}$ uses a set of parameters which are very close but not identical to the critic parameters which can provide stability in training. The target actor network $\mu_{\phi_{targ}}$ is used to compute an action which approximately maximizes $Q_{\phi_{targ}}$.

Using the target networks, the function to minimize for critic network training can be rewritten as:

$$L(\phi, D) = \underset{(s,a,r,s')\sim D}{E}\left[\left(Q_\phi(s,a) - \left(r + \gamma Q_{\phi_{targ}}\left(s', \mu_{\phi_{targ}}(s')\right)\right)\right)^2\right]. \qquad (12)$$

The entire process of generating data and training the four neural networks associated with DDPG is summarized in Figure 11.

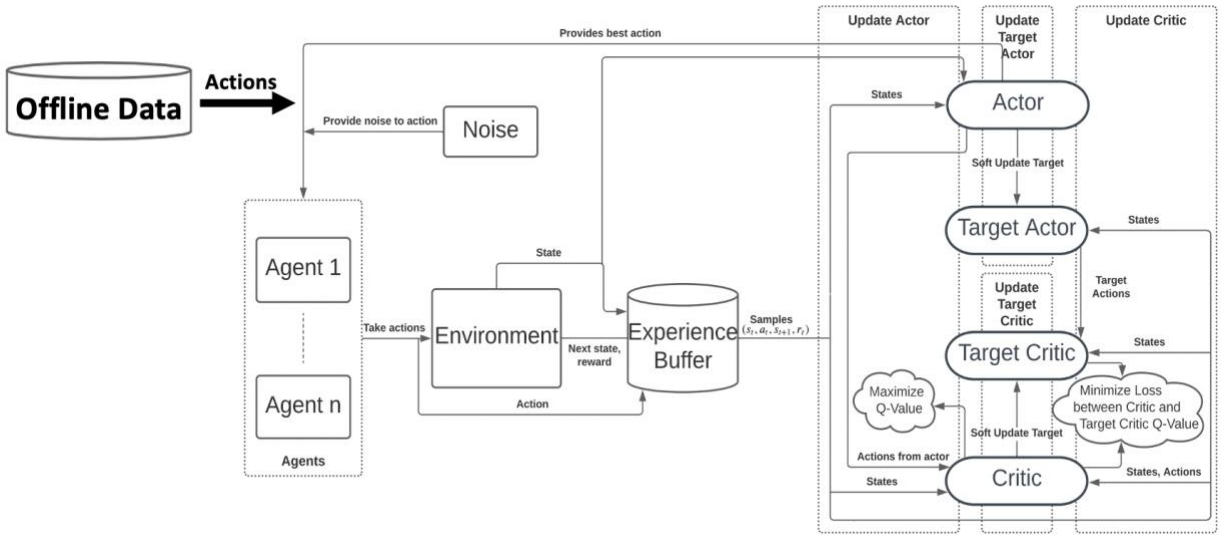


Figure 11*: DDPG Training and Data Generation Process with Offline Data*

Having designed the DDPG implementation, the neural networks were constructed using Keras (Chollet et al., 2015). Keras is a neural network library written in Python, that runs on top of a popular machine learning platform called Tensorflow (Abadi et al., 2016). The basic construction of the networks was derived from an already developed DDPG approach (Singh, 2020) with some minor modifications. In this approach, the actor network has two hidden layers with 256 units each that all use Rectified Linear Unit (relu) activation functions. The output layer has a number of units based on the number of actions and each unit uses a hyperbolic tangent (tanh) activation function. One modification made to this approach was changing the

activation function of the output layer to sigmoid instead of tanh. This choice for the output layer activation function was based on the desired range of the output. The coefficients outputted by networks need to be in the range $[\frac{\pi}{2}, \frac{25\pi}{18}]$ for the 10 coefficients defining the knee joint trajectories and in the range $[0, \frac{2\pi}{3}]$ for the 10 coefficients defining the hip joint trajectories. Since the sigmoid function outputs values in the range [0,1], it is simpler to use than tanh which outputs values in the range [-1,1]. The output of the sigmoid for the hip and knee joints (s_h, s_k) were simply transformed to the correct range for the coefficients (θ_h, θ_k) using the following equations:

$$\theta_h = \left(\frac{25\pi}{18} - \frac{\pi}{2}\right) s_h + \frac{\pi}{2}, \qquad \theta_k = \frac{2\pi}{3} s_k. \tag{13}$$

In addition to modifying the activation function of the output layer, other modifications were also made. First, a batch normalization layer was added before the output layer to make the training more stable by re-centering and re-scaling the data. This can be helpful in the case of DDPG as the training can be very unstable. Using similar logic, the input to the neural network was also standardized to have mean 0 and variance 1.

Following the actor, the critic network can be defined. The states input initially passes through two layers with 16 and 32 nodes respectively and relu activation functions. The actions input initially passes through one layer with 32 nodes and relu activation functions. The output from each provides an input to a network with two hidden layers with 256 nodes each and relu activations and an output layer with 1 node which returns the final Q-value. No modifications were made to this initial setup of the critic network.

As noted previously, the target networks are time-delayed copies of the original networks, so they are defined as duplicates of the main actor and critic networks. For the optimization process for the networks, the Adam optimization algorithm was used. Adam is a

replacement optimization algorithm for stochastic gradient descent for training neural networks (Kingma and Ba, 2014). It is a good choice for DDPG since it is easy to configure, and it has the ability to handle noisy problems that may result in sparse gradients.

Having defined the networks and the optimization approach, the hyperparameters of the networks needed to be set. One hyperparameter is the target network update parameter which was used in (11). This parameter can take on values between 0 and 1, and values closer to 0 mean the target will have less of a delay from the original. For this approach, this parameter was set to 0.001. Another hyperparameter to tune is the discount factor when training the critic, shown in (10). The discount factor can vary from 0 to 1, and the value quantifies the importance of future rewards with 0 meaning only immediate rewards are considered. In this approach, the discount factor was set to 0.90. The final and perhaps most important hyperparameter for training the neural networks is the learning rate of each network. If a learning rate is too small, training will progress very slowly with very small updates. If a learning rate is too large, updates will be too large and may result in divergent behaviors. When testing the actor network, using the default learning rate of 10^{-4} resulted in the rewards plot shown in Figure 12.

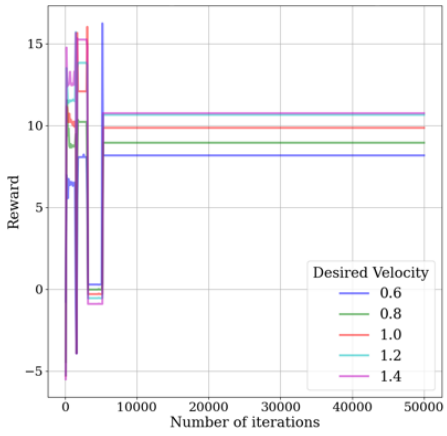


***Figure 12**: DDPG Training Failure with Default Learning Rate*

As is shown in the plot, updates occurred rapidly and then DDPG reached a poor solution. The rapid updates caused the values prior to the final network layer to approach high magnitude values. When these high values were passed to the sigmoid function, it returned values near the 0 and 1 bounds. An image of the sigmoid function is shown in Figure 13 to illustrate why inputs with high magnitude result in outputted values near 0 or 1.

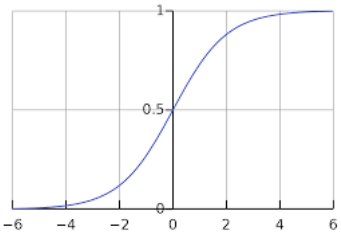


***Figure 13**: Visualization of sigmoid function to illustrate how large magnitude inputs result in outputs near 0 and 1 bounds*

In addition, network updates are based on the derivative of the activation function. So, in this case, the network will not be able to recover since the derivative and resulting updates will be minor when the function inputs have high magnitude. Ultimately, after many test trials, the learning rates of $5 * 10^{-6}$ and 10^{-4} were found to be functional for the actor and the critic respectively.

2.4) Evaluation of Off-Policy Reinforcement Learning Methods

Following the training of the policy, various metrics are used to evaluate the RL method. First, a plot of the rewards for each episode during training can be analyzed. With the reward function defined above, the maximum reward is 300 since there are a maximum of 300 actions (max number of simulation steps divided by simulation steps between each action) taken in an episode and each action can generate a reward up to 1. Ideally, the rewards should increase over time and converge close to the maximum reward value. However, analyzing the reward in this

manner is challenging with DDPG if the actions are highly sensitive to noise or the setup of each episode is different. Both situations apply for this project as the coefficient outputs of the neural network are highly sensitive to noise and each episode has a different desired velocity. Because of this, the rewards will be analyzed by designating 5 out of every 100 episodes as evaluation episodes. In these episodes, the policy will not update, no noise will be provided, and velocities of 0.6, 0.8, 1.0, 1.2, and 1.4 m/s will be tested. These results can be plotted, and the training process can be analyzed.

In addition to the rewards, the Q-values generated in the actor network training process can also be analyzed. The actor is trained to optimize the Q-values returned by the critic when the critic is provided the state and the action returned from the actor as input. In general, if the critic is successful, the Q-values should follow a similar trend as the rewards. If this is not the case, it is a strong indicator that the critic network training process is not working as intended.

Following the analysis of the rewards and Q-values, the final policy still needs to be evaluated on its ability to track desired velocities. This is done by plotting the policy's ability to track desired velocities of 0.6, 0.8, 1.0, 1.2, and 1.4 m/s and comparing the result to a known working on-policy method. The on-policy method used for comparison will be the approach developed in Castillo et al. (2019). Lastly, the number of episodes required to reach the final policy will be compared for each approach to determine the data efficiency of the method.

Chapter 3. Off-Policy RL for Bipedal Robot Velocity Tracking

Following the development of the off-policy approach framework, the approach was tested on its ability to successfully track various velocities. As was previously discussed, this provided a meaningful first test as velocity tracking had been solved for the RABBIT robot by on-policy approaches but not by off-policy approaches. The desired outcome of this test was that the approach would result in velocity tracking similar to the on-policy approach but complete the training in fewer episodes. First, the developed approach with offline data was compared to the basic DDPG method to demonstrate the benefits of offline data. After this, the approach was compared to an on-policy approach in terms of the preciseness of the velocity tracking and number of training episodes to converge to a high reward.

3.1) Basic DDPG for Bipedal Locomotion

To provide a baseline for the designed method, a basic DDPG approach was used. The code for this baseline was from a tutorial for DDPG which successfully solves the Inverted Pendulum environment (Singh, 2020). This code was then modified for the RABBIT environment. The state was redefined to a 15-tuple providing the robot's simulation state and the desired velocity. The robot's simulation state included the horizontal, vertical, and rotational positions of the hip joint, relative positions of the left and right hip and knee joints, and velocities of each joint. The actions were defined as a 4-tuple of the torques for each of the actuated joints. The reward function used was the reward function defined in (8). After redefining the states, actions, and rewards, training was performed for multiple different neural network hyperparameter configurations. The rewards and the Q-values from the best result are shown in Figure 14.

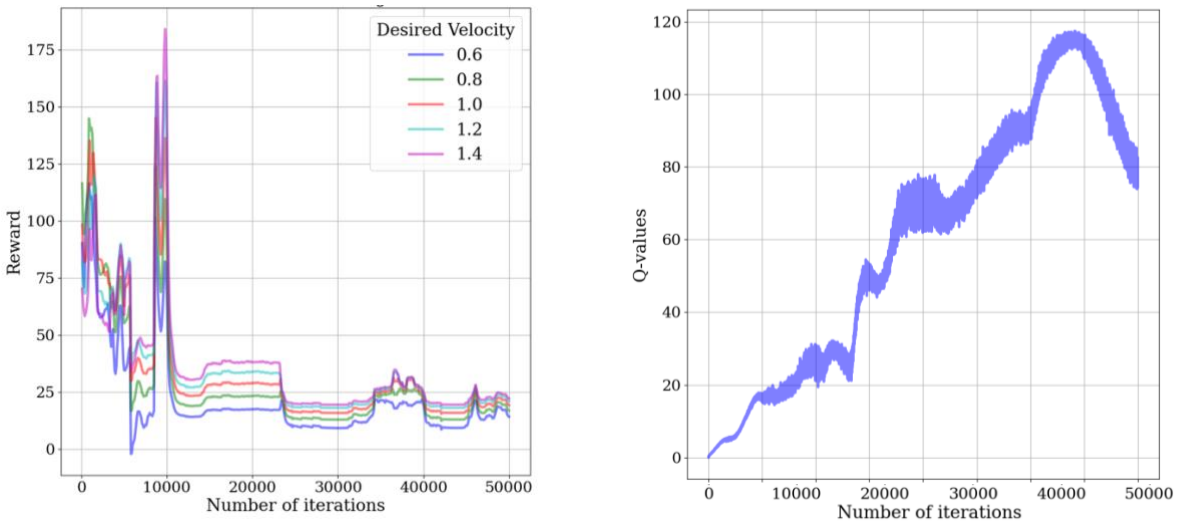


***Figure 14**: Rewards and Q-Values for Baseline DDPG Approach for Velocity Tracking*

As is shown in the rewards plot above, the maximum reward received was around 175 which was far less than the maximum potential reward of 3000. In addition, the rewards actually decreased with more training. As is shown in the Q-values plot, the Q-values did not trend the same way as the rewards, as the critic was never able to learn what actions generated high Q-values. When viewing the robot in simulation, it often fell immediately after just a few actions. Despite testing multiple configurations of hyperparameters, states, actions, and rewards, the basic DDPG method was never able to perform better than the result shown. Overall, it appeared that the basic DDPG approach was unable to adapt to the complexity of the bipedal walking movement required for the RABBIT robot.

3.2) Evaluation of State, Action, and Reward Space Transformations

After establishing that a basic DDPG method would fail to learn ideal actions for the RABBIT robot, the revised DDPG approach developed in Chapter 2. Designing Off-Policy RL Method was tested. In order to test if there was any benefit to offline data, the first test used no offline data, and the resulting rewards and Q-values from the test are shown in Figure 15.

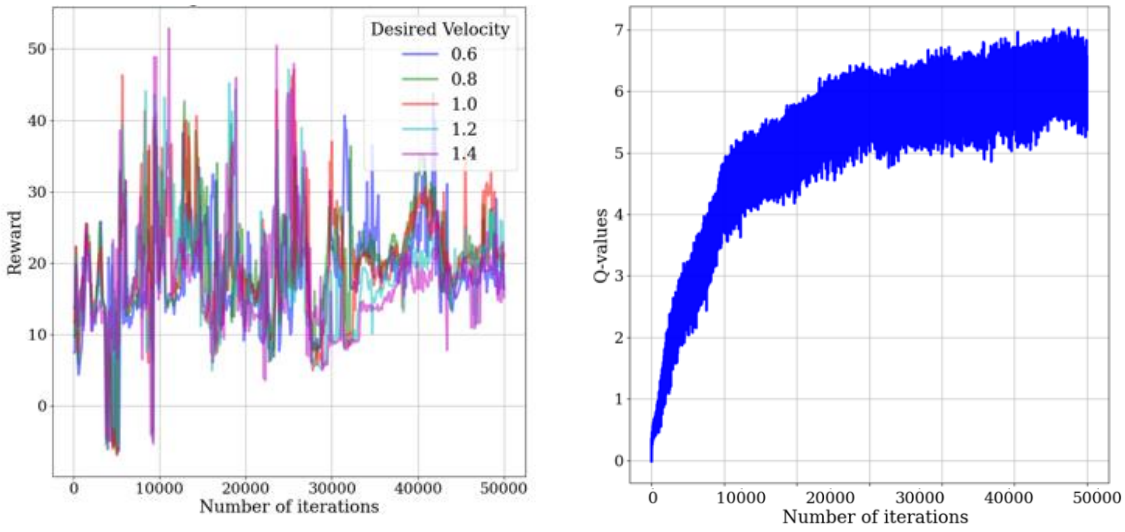


***Figure 15**: Rewards and Q-Values for revised DDPG without Offline Data for Velocity Tracking*

The policy only reached a reward around 50 which was significantly less than the maximum reward of 300. In addition, the Q-values saw a significant upward trend that was not matched by the rewards. When watching the robot simulations, the robot would walk for a bit and then fall. It never reached the maximum number of steps in an episode. However, this was still a significant improvement from the original method which resulted in the robot immediately falling and produced rewards and Q-values that actually decreased with more training.

Before adding the offline data to the current approach, there were other ways to consider improving the policy. Based on the robot simulations, the robot had issues not falling in addition to not being able to track velocities. To counter the falling, height was added to the reward function in order to punish actions that resulted in the height exiting an ideal height range. For the RABBIT robot, the ideal hip height is between 0.7 and 0.8 meters. Using these values, a height reward function R_h utilizing the current height h and the height prior to the action h_o was developed as shown:

$$R_h(h_c, h_o) = 0.5 * \mathrm{I}(|h_c - 0.75| < 0.05) - 0.5 * \mathrm{I}(|h_0 - 0.75| < 0.05) * \mathrm{I}(|h_c - 0.75| > 0.05). \quad (14)$$

This reward function provides a reward of -0.5 if the height was previously in the correct range before the action and is currently not in the range. It also provides a reward of 0.5 if the current height is in the correct range. This height reward R_h can be added to the original reward from (8), here defined as R_v, with average velocity since last action v_a and desired velocity v_d to get a new reward function R_f:

$$R_f(v_a, v_d, h_c, h_o) = R_v(v_a, v_d) + R_h(h_c, h_o). \quad (15)$$

The resulting rewards and Q-values from training a policy with the updated reward function are shown in Figure 16.

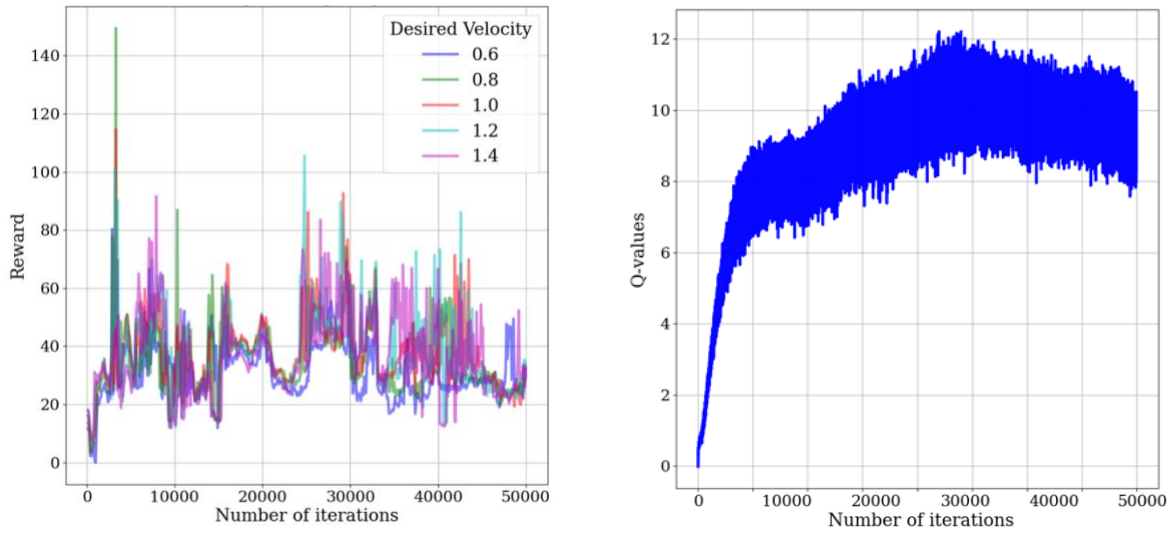


***Figure 16**: Rewards and Q-Values for revised DDPG with Height Reward for Velocity Tracking*

Using height, the maximum reward obtained by the policy in training was 140 which was significantly less than the new maximum theoretical reward of 450. The Q-values results were comparable to when height was not utilized. Overall, it appeared that height slightly improved the rewards, so for the following tests, the reward function with height was used.

3.3) Utilizing of Offline Action Data in Training

For using the offline data in training, tests for using both one set and ten sets of coefficients to generate data were performed. The one set of coefficients was generated from FROST with a desired velocity of 1. The resulting rewards and Q-values from training a policy with one set of coefficients are shown in Figure 17.

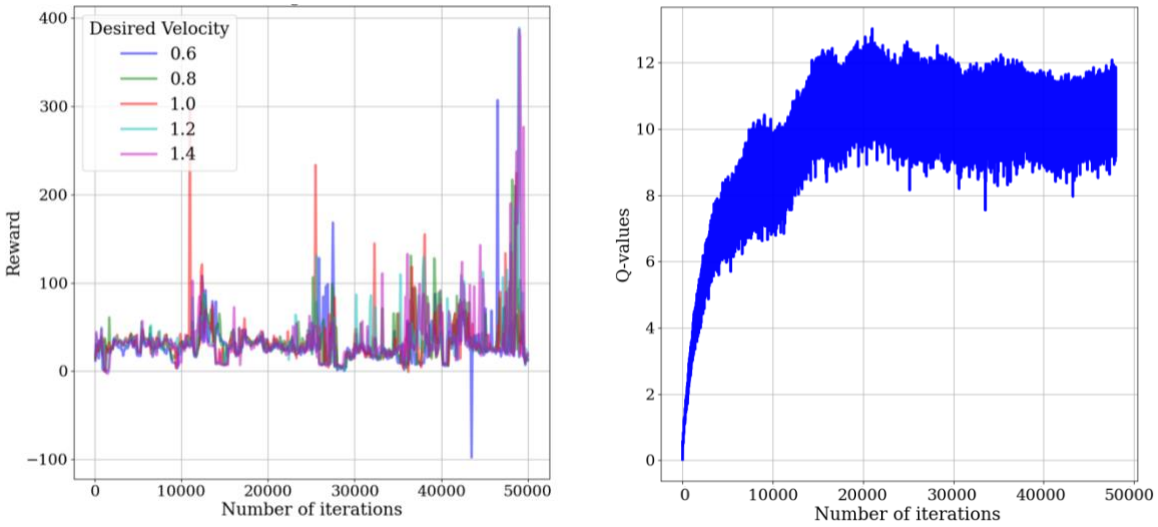


***Figure 17**: Rewards and Q-Values for revised DDPG with One Offline Action for Velocity Tracking*

Given that the maximum reward was 450, rewards near 400 exhibited an improved policy when compared to policies that were developed not using offline data. This showed that using even a small amount of offline data was helpful in developing a working policy. However, the Q-values did not appear to follow the trends of the rewards as they actually slightly decreased towards the end of training. Only using one known action limited the critic network's ability to learn a variety of good actions. For the next test, ten sets of coefficients generated from FROST with desired velocities of 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4, and 1.5 were used. The resulting rewards and Q-values from training a policy with the ten sets of coefficients are shown in Figure 18.

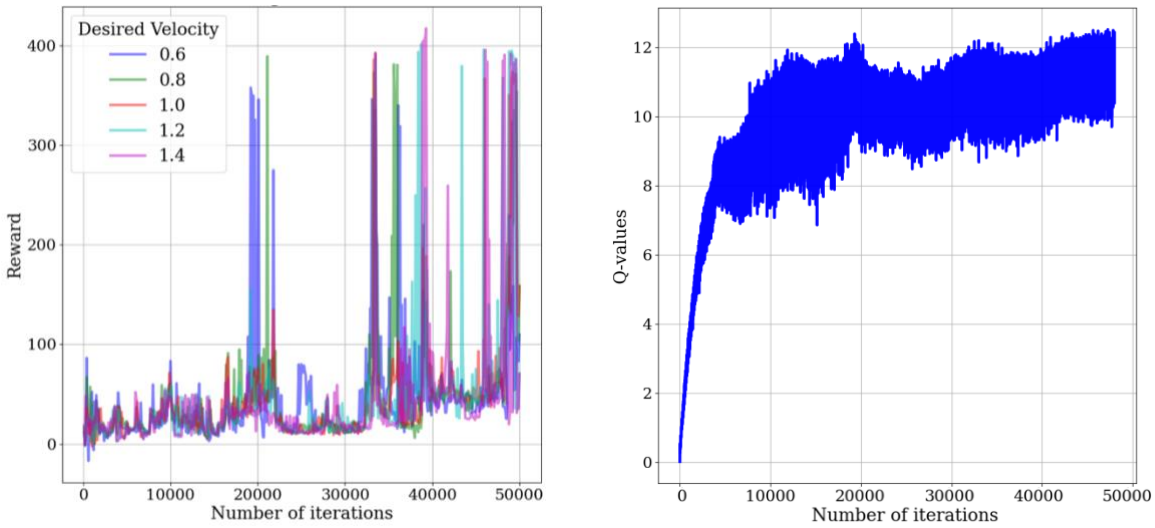


***Figure 18**: Rewards and Q-Values for revised DDPG with Ten Offline Actions for Velocity Tracking*

Using more offline data allowed for more consistently high rewards. In addition, the Q-values matched the rewards far better than when using one set of coefficients. Having developed a policy that exhibited high rewards, the next assessment was to see whether these high rewards resulted in precise velocity tracking. The policy's ability to track desired velocities of 0.6, 0.8, 1.0, 1.2, and 1.4 m/s was plotted (shown in blue) and compared to a known on-policy method (shown in green) in Figure 19.

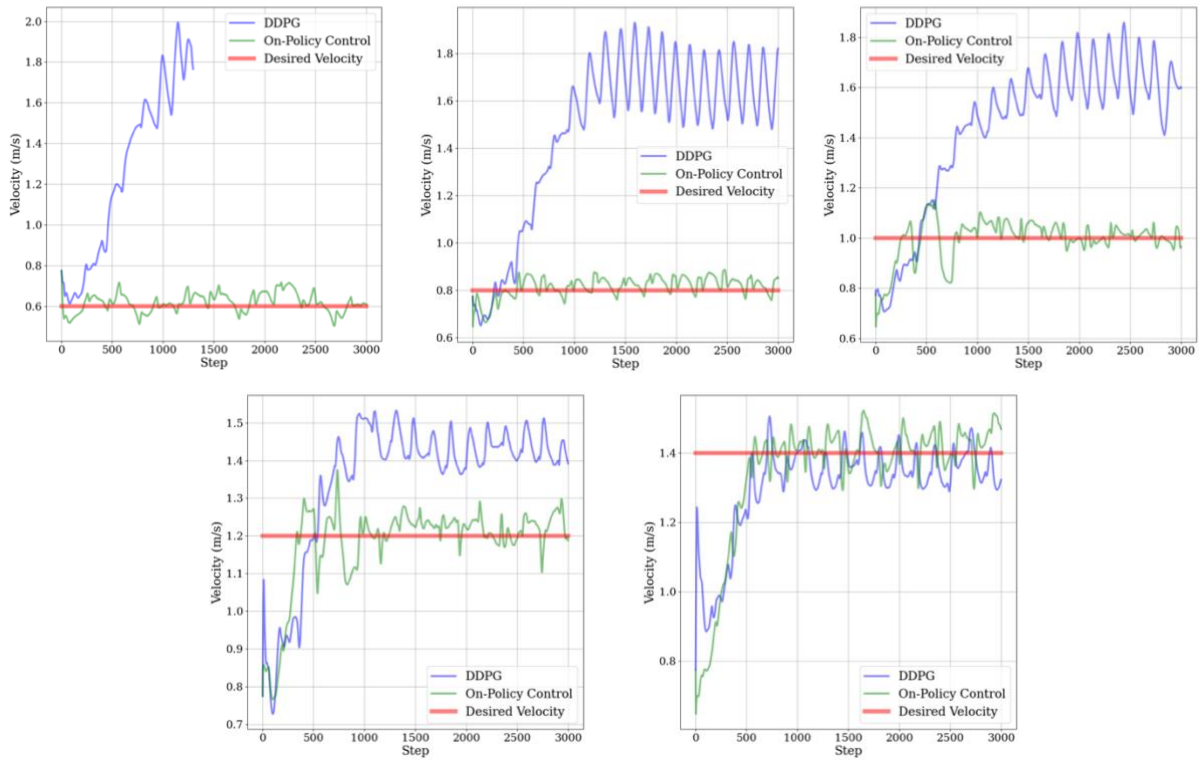


***Figure 19**: Velocity Tracking Performance for initial revised DDPG method*

With the exception of the results from a desired velocity of 0.6, the bipedal robot learned a policy that resulted in stable walking, as the robot reached the maximum number of steps without falling. However, the biggest issue was the policy was unable to develop walking movements that would track the desired velocities. The policy placed a higher weight on not falling than dialing in on the correct velocity. This was a result of how the reward function was defined, and a more ideal reward function would need to provide more reward to actions that resulted in velocities that were near the desired velocity.

3.4) Balancing Velocity Tracking and Walking Stablility in Reward Defintion

Due to the results above, the reward function was revised to provide higher rewards for actions that resulted in velocities near the desired velocity. This was done by providing a small

reward when the velocity was somewhat close to the desired velocity and then a higher reward when it was very close. The additional term, R_d, that was used to accomplish this is shown:

$$R_d(v_a, v_d) = \mathrm{I}(|v_a - v_d| < 0.1) + \mathrm{I}(|v_a - v_d| < 0.05). \quad (16)$$

This new reward term was added to the previous reward shown in (15) to obtain the following revised reward function:

$$R_f(v_a, v_d, h_c, h_o) = R_v(v_a, v_d) + R_h(h_c, h_o) + R_d(v_a, v_d). \quad (17)$$

A new test was performed with this revised reward function, and the results of the test in terms of the rewards and Q-values are shown in Figure 20.

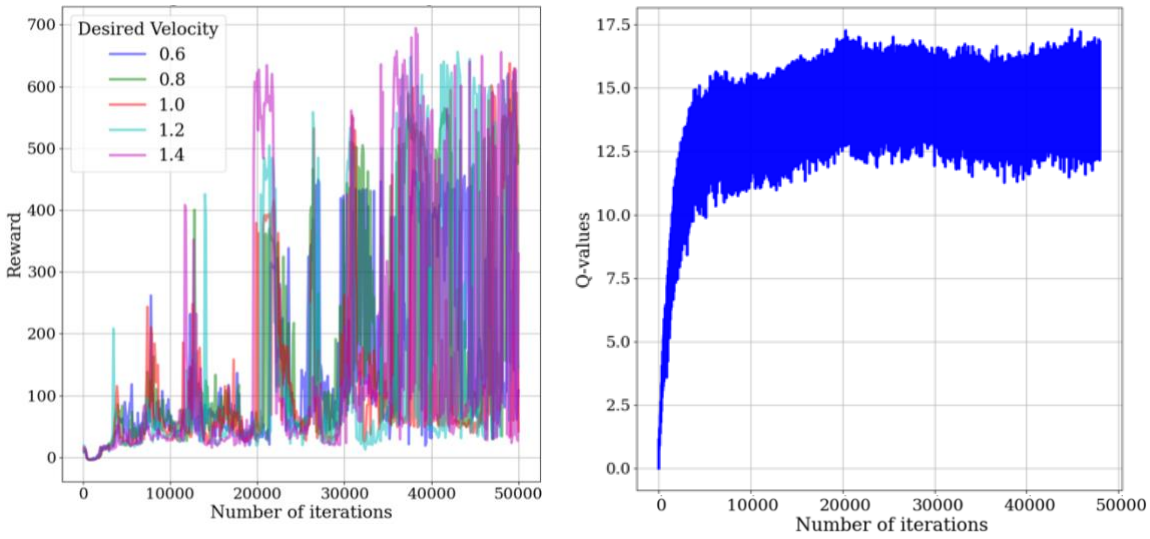


***Figure 20**: Rewards and Q-Values for final revised DDPG for Velocity Tracking*

Revising the reward function allowed the policy to start obtaining high rewards around 20,000 episodes of training and to consistently return high rewards for all velocities after 40,000 episodes of training. In addition, the trend of Q-values closely resembled the rewards during training. The results of the policy tracking desired velocities of 0.6, 0.8, 1.0, 1.2, and 1.4 m/s compared to the on-policy method is shown in Figure 21.

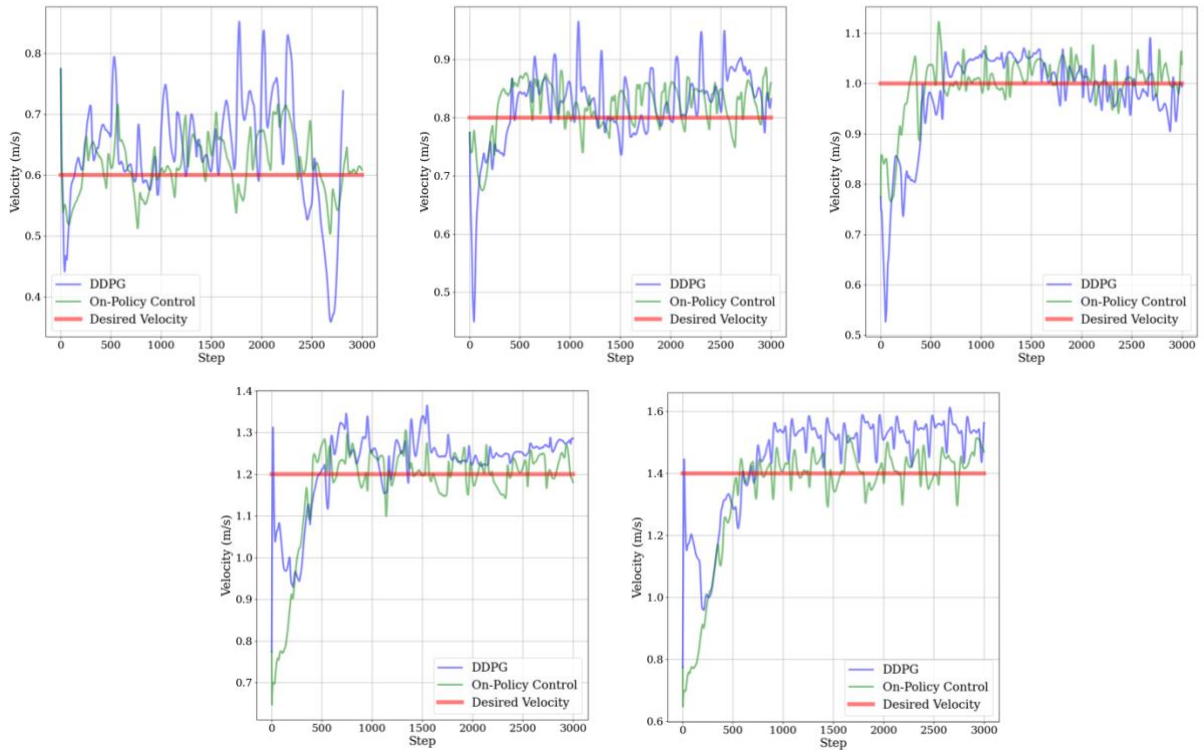


***Figure 21**: Velocity Tracking Performance for final revised DDPG method*

From the velocity plots, it is evident that the reward adjustment resulted in a policy that performed much better for velocity tracking. Stable walking motion was maintained for every desired velocity, and velocities were very close to the desired values. Although the policy's velocity deviates slightly from the desired values for velocities of 0.6 and 1.4 m/s, the results are mostly comparable to the on-policy baseline. In addition, the developed off-policy method was far more data efficient than the on-policy method. The off-policy method required just 49,600 episodes to achieve velocity tracking, while the on-policy baseline required 234,500 episodes. In addition, the off-policy method needed around 20,000 episodes for the policy to produce stable walking motions, while the on-policy baseline needed around 100,000 episodes.

Chapter 4. Off-Policy RL for Bipedal Robot Sloped Terrain Locomotion

4.1) Utilization of Off-Task Data

In Chapter 3, it was illustrated that a popular off-policy RL method, DDPG, could be employed for robot velocity tracking by using a small offline dataset and information from the underlying physics of robot locomotion. The approach performed velocity tracking comparably to popular on-policy methods and was more data efficient. However, the velocity tracking problem was already solved using on-policy RL. While this improved data efficiency is certainly beneficial, the most important potential benefit of off-policy RL is its ability to use a large previously collected dataset to enable robotic tasks that have not been obtained by on-policy RL methods. The process of training the robot to track desired velocities generated a large dataset of actions, and an off-policy RL approach has potential to utilize this data to train other robotic tasks. An example task can be to teach a robot to walk up and down hills of varying slopes. The environment used to train the RABBIT robot to complete this task is shown in Figure 22.

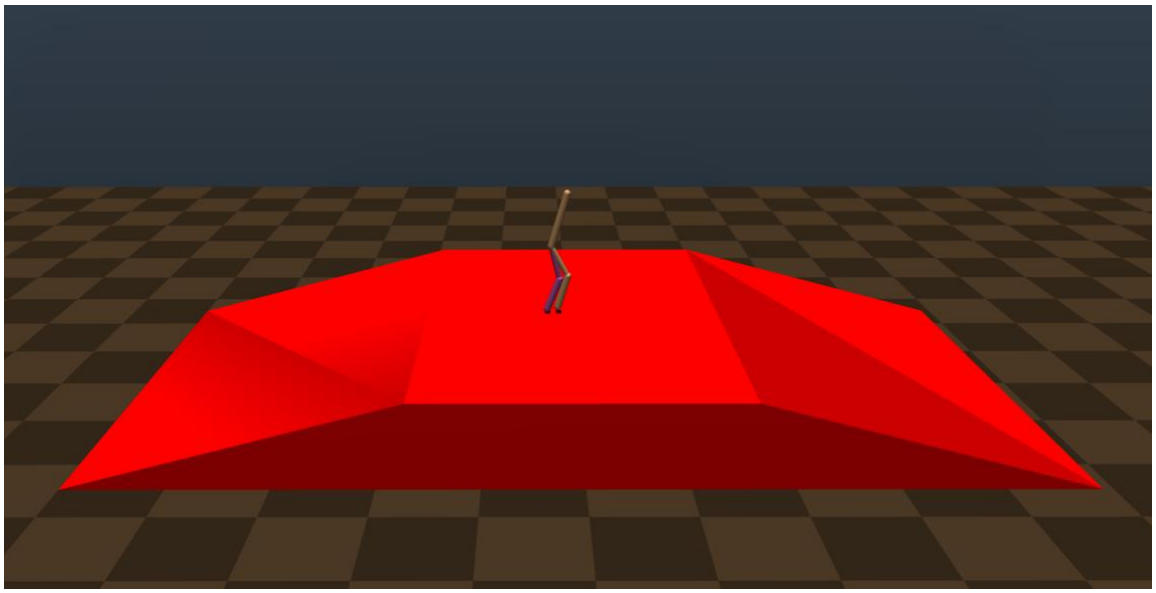


***Figure 22**: Environment for Sloped Terrain Test*

For developing the large static dataset, the action data generated from teaching a bipedal robot to track velocities was used. The action data from the on-policy method (Castillo et al., 2019) and the previously implemented DDPG method were added to the dataset. A dataset of over 4000 sets of different coefficients was obtained for this approach. During training, this dataset was randomly sampled, and the actions sampled were taken by the bipedal robot to generate data for the experience buffer.

4.2) Alterations to Developed RL Method

For training the robot to walk up sloped terrain, a few modifications were made to the method used for velocity tracking. First, the reward function was changed, since the robot's goal was now to walk up and down hills without falling. To achieve this goal, the robot needed to walk the entire distance of the environment without falling. When defining the reward, distance traveled when a given action was taken d_c and total distance traveled for the entire episode d_t were used to make sure the robot was successfully getting to the other side of the hill. For maintaining robot stability, the percentage of simulation steps taken (number of simulation steps prior to falling divided by the maximum number of steps) n and whether the robot fell f were used. For additional stability, increased reward was provided if the torso angle, a, was within a desired range of -0.02 to 0.38 degrees. Lastly, to prevent the robot from running up the hill or barely moving the reward was reduced if the robot's forward velocity v was outside the range of 0.5 to 2.0 m/s. After testing reward functions with varying weights for each variable, the reward function shown in (18) was ultimately employed.

$$R(d_c, d_t, f, n, a, v) = \frac{d_t n}{6}(5d_c I(\bar{f}) - I(f)) + 0.5I(|a - 0.18| < 0.2) - 0.5I(|v - 1.25| > 0.75) \quad (18)$$

In addition to changing the reward function, the state function was also changed. Previously the desired velocity, velocity of the last 200 simulation steps, and the height of the robot hip were used as they were all crucial for velocity tracking. However, for walking up hills, other factors like positions of the various joints are more important, as the robot must be properly angled to walk up each hill. Therefore, the horizontal, vertical, and rotational positions of the hip joint (x, y, q_t), relative positions of the left and right hip and knee joints $(q_{sh}, q_{nsh}, q_{sk}, q_{nsk})$, and velocities of each joint $(v_x, v_y, v_{q_t}, v_{q_{sh}}, v_{q_{nsh}}, v_{q_{sk}}, v_{q_{nsk}})$ were considered. In addition, the slope of the hill m can provide useful information for walking up hills. All of these attributes were included in the state definition except for the horizontal position of the hip joint. The issue with using this attribute is that the policy may associate certain parts of the hill with certain horizontal locations of the hip joint. If the horizontal position of the hill within the environment is changed slightly, the policy would no longer function as desired. Thus, the state was defined as $(y, q_t, q_{sh}, q_{nsh}, q_{sk}, q_{nsk}, v_x, v_y, v_{q_t}, v_{q_{sh}}, v_{q_{nsh}}, v_{q_{sk}}, v_{q_{nsk}}, m)$.

Training for sloped terrain also required many different configurations. First, the robot needed to learn to walk both up and down a given hill. If the robot always began training at the bottom of the hill, it would likely not reach the top until after many episodes of training which would provide it limited training in walking downhill. To prevent this issue, the robot began 20% of the training episodes at the top of the hill to learn downhill walking. The robot also needed to learn how to walk on hills of varying slopes. Slopes of 7, 11, and 15 degrees were used in testing. The slope for each episode was randomly chosen, and the environment was updated to reflect the slope change. For testing how the policy was performing, the last 3 episodes of every 100 episodes had no noise and tested the policy's ability to walk up and down hills for each slope.

4.3) Implementing Sloped Terrain Locomotion without Offline Dataset

To investigate whether there were benefits for using an offline dataset in this problem, the approach was first implemented without offline data. The same DDPG algorithm developed for velocity tracking was used along with the revised reward function, state space, and training configurations described in the Alterations to Developed RL Method section above. A policy was trained for 50,000 episodes to see if it converged to an optimal policy for sloped terrain locomotion. Given the reward function defined above, an optimal policy would result in rewards near 1,300 for 7, 11, and 15 percent grades. The resulting rewards and Q-values during training without offline data are illustrated in Figure 23.

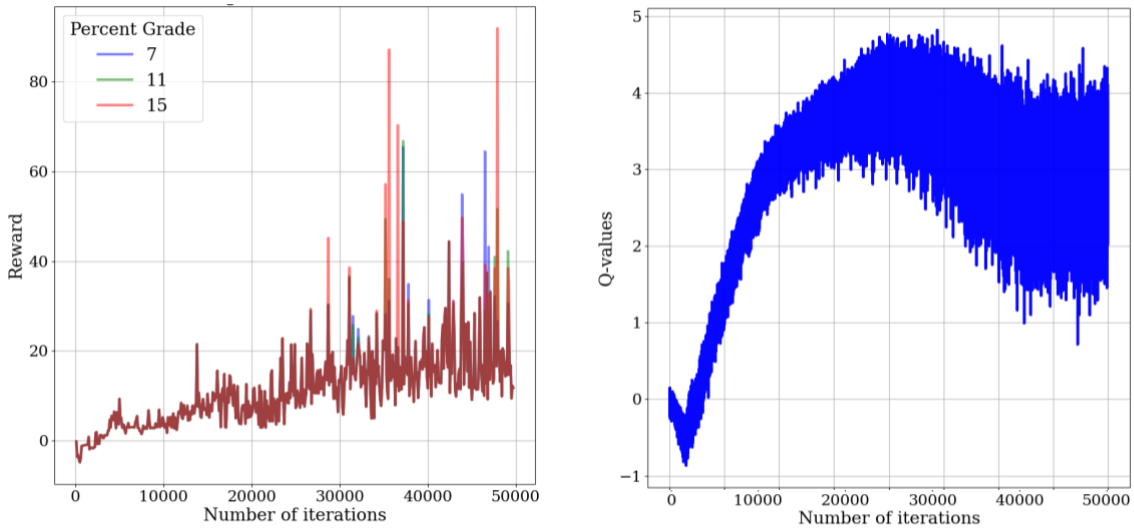


***Figure 23**: Rewards and Q-Values for Sloped Terrain Training Without Offline Dataset*

As is shown in the rewards plot above, the reward rarely reached values above 50 which was far less than the ideal reward of 1,300. The plot also shows that during most of the training process the returned rewards were the same for each grade value, which means that the robot mostly fell before reaching the slope. The fact that this method was barely able to learn how to walk on flat ground in 50,000 episodes of training exhibits that it is not functional for training a robot to walk up hills. In addition, the trend of Q-values did not match the trend of rewards. The

Q-values actually decreased after 25,000 episodes of training while the rewards increased. Overall, this displayed that without any offline data, the critic was unable to learn which actions were optimal in given states.

4.4) Implementing Sloped Terrain Locomotion with Offline Dataset

Having demonstrated the potential issues with training the critic network without any offline data, the same approach was tested except with sampling from the offline dataset. The rewards and Q-values over the course of training are shown in Figure 24.

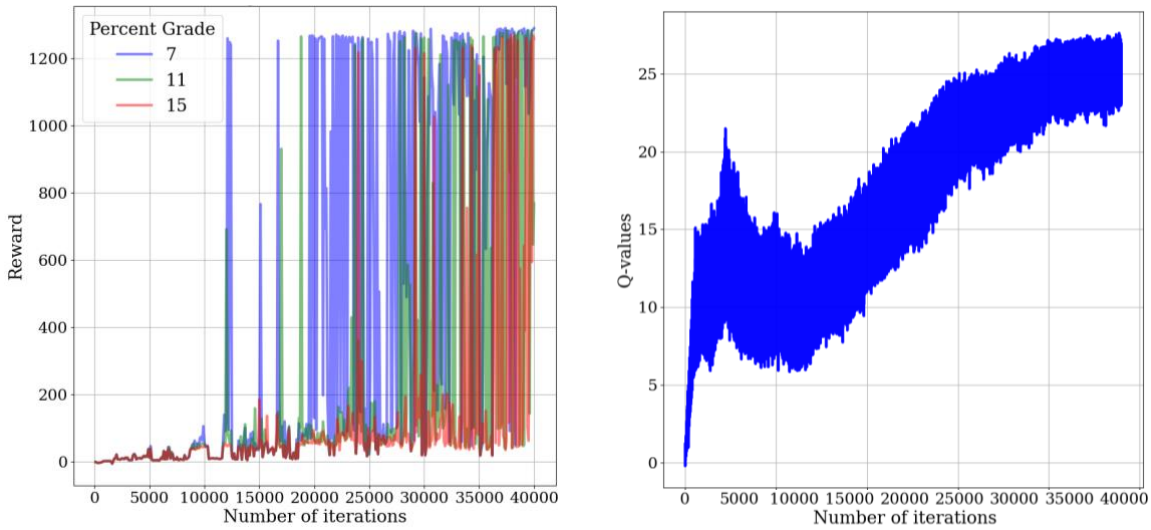
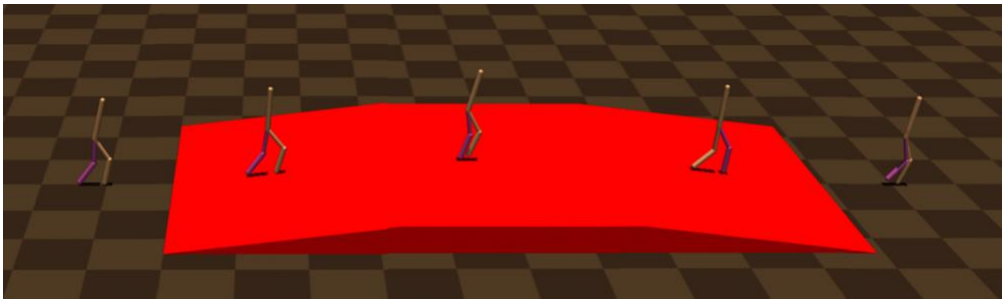


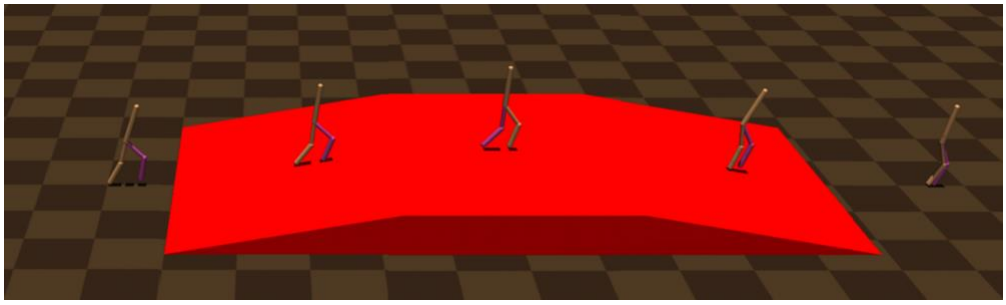
***Figure 24**: Rewards and Q-values for Sloped Terrain Training with Offline Dataset*

As is shown in the rewards plot above, using offline data allowed the policy to learn action sequences that resulted in rewards near the optimal value of 1,300. In addition, the method was relatively efficient as it started to return high rewards after around 12,000 episodes and only required 40,000 episodes to reach a consistent optimal solution. Along with the actor network generating actions with high rewards, the critic network also learned successfully as the trend of Q-values closely resembled the trend of rewards. Having exhibited desirable rewards and Q-values, it was important to examine whether the high reward actions resulted in stable

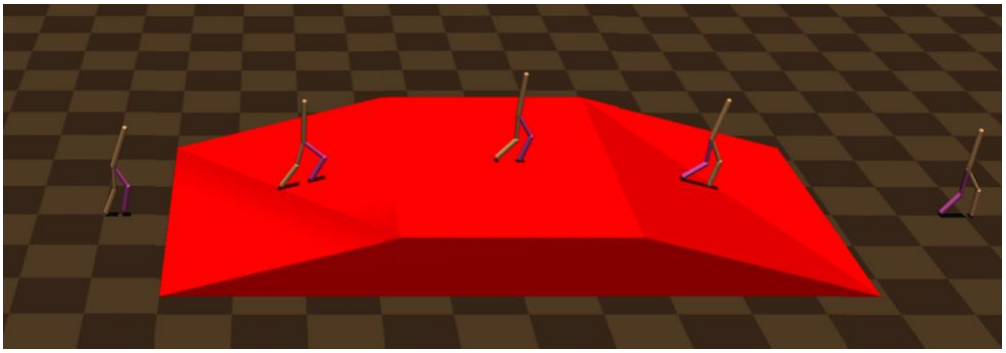
walking for the RABBIT robot on sloped terrain. A visualization of the RABBIT robot's motion when it followed the trained policy on 7, 11, and 15 degree slopes is shown in Figure 25.



7° slope



11° slope



15° slope

***Figure 25**: Motion produced by trained policy for 7°, 11°, and 15° slopes*

There were a few interesting observations that were drawn from the robot motion on the sloped terrain. The first observation was the robot used a completely different walking gait for walking uphill when compared to walking on flat ground. The robot used a high knee drive to propel itself up the hill. In addition, on larger slopes, it lengthened it stride to propel itself up the hill. This result was encouraging as there were no walking gaits like this in the offline dataset.

The robot was able to use the offline data to learn stable walking motions. Once it learned these actions as a solid baseline, the robot was able to explore actions that slightly deviated from this baseline, and the robot ultimately was able to learn the best walking motions for going uphill. A similar process was evident with downhill walking. When walking downhill, the robot walked slowly and cautiously and occasionally dragged its feet to prevent itself from falling. Again, these actions were not in the offline dataset, but the robot was able to learn them by using the offline dataset as a baseline. A final observation was that the policy adapted well to changes in terrain. In order to not fall, the robot needed to generate enough momentum to get up the hill, slow itself as it reaches the top, transition to a stable walking motion on a flat area, slow itself again to not fall down the hill, and then transition back to a stable walking motion at the bottom. Having a policy that adapted to these sudden changes illustrated the value in using a large and diverse dataset in RL policy training.

Chapter 5. Conclusions

5.1) Contributions

In this thesis, offline data was used to make an off-policy RL algorithm functional for bipedal locomotion tasks. To the best of my knowledge, this was the first time an off-policy RL approach had been successful in training a bipedal robot to track desired velocities and walk-up hills of varying slopes. In the case of velocity tracking, the main contribution was training a policy to complete this task in a more data efficient manner than on-policy approaches by using a small offline dataset. In the case of sloped terrain locomotion, the main contribution was using a large diverse dataset from the velocity tracking training to train a robot to complete this task. This was a significant result given that both traditional on-policy and off-policy methods were too data inefficient to reasonably train this task.

5.2) Additional Applications

There are a wide variety of additional applications for this work. In the bipedal locomotion setting, this work illustrates a paradigm for developing policies for varying locomotion tasks. Once a policy for a given task is developed, this data can be utilized to help train other tasks. Over the course of training many different tasks, a large and diverse dataset can be developed that can be instrumental in developing data efficient RL approaches with generalizable policies.

Although this project outlined specific methods for using offline data in bipedal locomotion, using offline datasets in conjunction with off-policy RL can have applications in any field using RL. RL approaches have been attempted in the fields of autonomous driving, healthcare, and natural language with mixed success. While each field presents completely

different challenges, the use of offline data can be helpful in developing functional RL approaches.

5.3) Future Work

Due to the time constraints of this project and the wide scope of RL, there is a wide array of future research that can be developed from this project. First, the results from this work can be further tuned. For example, the developed approach for velocity tracking resulted in the robot velocity occasionally deviating from the desired value for certain desired velocities. Exploring different hyperparameter values and reward functions are areas of future work for improving velocity tracking. In addition, future work should examine ways for obtaining a larger dataset for velocity tracking as the dataset used in this approach had limited data. Further studies should also investigate testing the approach on robots with more complex mechanical structures than RABBIT as these robots are more likely to be useful for real world applications. Another potential area of future research would be using the developed off-policy RL approach to train a robot to transition from simulation to the real world. This could involve using simulation data to help train a policy for robot locomotion in the real world.

5.4) Summary

This research focused on developing an off-policy RL approach that could train generalizable RL policies for bipedal locomotion tasks in a data efficient manner. The final approach accomplished this by considering the underlying physics of robot walking when defining the robot actions, using customized reward functions and robot states, and exploiting offline data during policy training. For the velocity tracking task, a small dataset of ten known

actions was used to develop a policy that achieved comparable velocity tracking to a known on-policy approach with less than one-fourth the training episodes. The data from the velocity tracking training process was then provided as offline data to help train a robot to walk up and down hills of varying grade. By using an offline dataset of actions generated from velocity tracking, a policy was able to be trained to complete this task in just over 30,000 episodes. This marked a significant improvement over methods not using offline data which had been too data inefficient to train this task. The results from this research were valuable in providing data efficient RL approaches for training bipedal locomotion tasks and exhibiting the value of offline datasets for developing functional RL methods.

Appendix A: Pseudocode for Popular RL Algorithms

Algorithm 1 On-policy policy gradient with Monte Carlo estimator

initialize θ_0
for iteration $k \in [0, \ldots, K]$ **do**
  sample trajectories $\{\tau_i\}$ by running $\pi_{\theta_k}(\mathbf{a}_t|\mathbf{s}_t)$ ▷ each τ_i consists of $\mathbf{s}_{i,0}, \mathbf{a}_{i,0}, \ldots, \mathbf{s}_{i,H}, \mathbf{a}_{i,H}$
  compute $\mathcal{R}_{i,t} = \sum_{t'=t}^{H} \gamma^{t'-t} r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$
  fit $b(\mathbf{s}_t)$ to $\{\mathcal{R}i, t\}$ ▷ use constant $b_t = \frac{1}{N}\sum_i \mathcal{R}i, t$, or fit $b(\mathbf{s}_t)$ to $\{\mathcal{R}i, t\}$
  compute $\hat{A}(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) = \mathcal{R}_{i,t} - b(\mathbf{s}_t)$
  estimate $\nabla_{\theta_k} J(\pi_{\theta_k}) \approx \sum_{i,t} \nabla_{\theta_k} \log \pi_{\theta_k}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \hat{A}(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$
  update parameters: $\theta_{k+1} \leftarrow \theta_k + \alpha \nabla_{\theta_k} J(\pi_{\theta_k})$
end for

***Figure 26**: Policy Gradients Pseudocode (Levine et al., 2020)*

Algorithm 2 Generic Q-learning (includes FQI and DQN as special cases)

initialize ϕ_0
initialize $\pi_0(\mathbf{a}|\mathbf{s}) = \epsilon \mathcal{U}(\mathbf{a}) + (1-\epsilon)\delta(\mathbf{a} = \arg\max_{\mathbf{a}} Q_{\phi_0}(\mathbf{s}, \mathbf{a}))$ ▷ Use ϵ-greedy exploration
initialize replay buffer $\mathcal{D} = \emptyset$ as a ring buffer of fixed size
initialize $\mathbf{s} \sim d_0(\mathbf{s})$
for iteration $k \in [0, \ldots, K]$ **do**
  for step $s \in [0, \ldots, S-1]$ **do**
    $\mathbf{a} \sim \pi_k(\mathbf{a}|\mathbf{s})$ ▷ sample action from exploration policy
    $\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ ▷ sample next state from MDP
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}, \mathbf{a}, \mathbf{s}', r(\mathbf{s}, \mathbf{a}))\}$ ▷ append to buffer, purging old data if buffer too big
  end for
  $\phi_{k,0} \leftarrow \phi_k$
  for gradient step $g \in [0, \ldots, G-1]$ **do**
    sample batch $B \subset \mathcal{D}$ ▷ $B = \{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_t)\}$
    estimate error $\mathcal{E}(B, \phi_{k,g}) = \sum_i \left(Q_{\phi_{k,g}} - (r_i + \gamma \max_{\mathbf{a}'} Q_{\phi_k}(\mathbf{s}', \mathbf{a}'))\right)^2$
    update parameters: $\phi_{k,g+1} \leftarrow \phi_{k,g} - \alpha \nabla_{\phi_{k,g}} \mathcal{E}(B, \phi_{k,g})$
  end for
  $\phi_{k+1} \leftarrow \phi_{k,G}$ ▷ update parameters
end for

***Figure 27**: Generic Q-Learning Pseudocode (Levine et al., 2020)*

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

***Figure 28**: Original DDPG Algorithm (Lillicrap et al., 2015)*

Appendix B: Project Code and Additional Resources

Project Code:

The project code can be found at the GitHub link below. Note that this code requires the MuJoCo physics engine to run. MuJoCo requires a license, and instructions for obtaining a license can be found at: https://www.roboti.us/license.html.

https://github.com/ballas13/Off-Policy-RL-Project-Bipedals

Additional Resources:

For more information and tutorials regarding some of the topics discussed in this project, see the following online resources.

Machine and Deep Learning Fundamentals Tutorial from Deep Lizard:
https://deeplizard.com/learn/playlist/PLZbbT5o_s2xq7LwI2y8_QtvuXZedL6tQU

Machine Learning Course from Udacity
https://www.udacity.com/course/machine-learning--ud262

Reinforcement Learning Tutorial from OpenAI
https://spinningup.openai.com/en/latest/

Reinforcement Learning Tutorial from Deep Lizard
https://deeplizard.com/learn/playlist/PLZbbT5o_s2xoWNVdDudn51XM8lOuZ_Njv

Reinforcement Learning Course from Udacity
https://www.udacity.com/course/reinforcement-learning--ud600

Offline Reinforcement Learning Lecture by Sergey Levine
https://www.youtube.com/watch?v=qgZPZREor5I

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zheng, X. (2016). Tensorflow: A system for large-scale machine learning. Google Brain, 2016. *arXiv preprint*.

Achiam, J. (2018). Spinning Up in Deep Reinforcement Learning. *GitHub, GitHub repository,* 2018.

Agarwal, R., Schuurmans, D., and Norouzi, M. (2019). An optimistic perspective on offline reinforcement learning. arXiv preprint arXiv:1907.04543.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). OpenAI Gym. *ArXiv e-prints*, June 2016.

Castillo, G., Weng, B., Hereid, A. and Zhang, W. (2019). Reinforcement learning meets hybrid zero dynamics: a case study for RABBIT. *IEEE International Conference on Robotics and Automation (ICRA), 2019*.

Chollet, F. et al. (2015). Keras. *GitHub, GitHub repository,* 2015.

Hereid, A. and Ames, A. (2017). Frost: Fast robot optimization and simulation toolkit. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS),* Sept 2017, pp. 719–726.

Janner, M., Fu, J., Zhang, M., and Levine, S. (2019). When to trust your model: Model-based policy optimization. *In Advances in Neural Information Processing Systems 32*, pp. 12498–12509.

Kalashnikov, D., Irpan, A., Pastor, P., Ibarz, J., Herzog, A., Jang, E., Quillen, D., Holly, E., Kalakrishnan, M., Vanhoucke, V., et al. (2018). Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293*.

Kim, D., Lee, J. and Sentis, L. (2017), Robust Dynamic Locomotion via Reinforcement Learning and Novel Whole Body Controller. *arXiv.org, 2017*.

Kingma, D. and Ba, J (2014). Adam: A method for stochastic optimization. Proceedings of *International Conference on Learning Representations (ICLR), abs/1412.6980, 2014*

Levine, S., Kumar, A., Tucker, G., and Fu, J. (2020). Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems. *arXiv preprint arXiv:2005.01643*.

Li, Y. (2018). Deep Reinforcement Learning: An Overview. *arXiv preprint arXiv:1701.07274*.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *ArXiv:1509.02971,* 2015.

Matheron, G., Perrin, N., and Sigaud O. (2019). The problem with DDPG: understanding failures in deterministic environments with sparse rewards. *arXiv preprint arXiv:1911.11679*, 2019.

Singh, H. (2020). Deep Deterministic Policy Gradient (DDPG). *GitHub, GitHub repository,* 2020.

Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct 2012, pp. 5026–5033.

Westervelt, E., Grizzle, J., Chevallereau, C., Choi, J., and Morris, B. (2007). Feedback Control of Dynamic Bipedal Robot Locomotion. *London, U.K.: Taylor & Francis*, 2007.

Xie, Z., Clary, P., Dao, J., Morais, P., Hurst, J., and Van de Panne, M. (2019). Iterative Reinforcement Learning Based Design of Dynamic Locomotion Skills for Cassie. *arXiv:1903.09537*, 2019.

Zhu, H., Yu, J., Gupta A., Shah, D., Hartikainen, K., Singh, A., Kumar, V., and Levine, S. (2019). The Ingredients of Real World Robotic Reinforcement Learning. Submitted to *IEEE International Conference on Robotics and Automation (ICRA), 2020.*